XML and the Text Encoding Initiative

by

Megan Winget

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science

Chapel Hill, North Carolina

December, 1999

Approved by:

Advisor

Megan Winget. XML and the Text Encoding Initiative. A Master's paper for the M.S. in I.S. degree. December, 1999. 46 Pages. Advisor: Charles Viles.

This paper examines how documents marked up using the Text Encoding Initiative (TEI) SGML guidelines might be automatically transformed to conform to Document Type Definitions (DTDs) in the Extensible Markup Language (XML). The paper provides an overview of the TEI guidelines, and the TEI-Lite subset. There is a survey of XML with particular attention paid to the proposed XML version of the TEI-Lite DTD. Preliminary experiments returned a 31% success rate, showing that the available tools that translate SGML TEI-Lite conformant documents to XML versions are flawed. These tools require significant manual intervention in order to function correctly. This paper concludes that if we can overcome the problems with translation, XML will become the chief means for transmitting and publishing texts on the web. Sources include publicly available web pages and standards body produced specifications.

Headings:

XML (Document markup language)

Text Encoding Initiative

SGML (Document markup language)

University of North Carolina at Chapel Hill. Documenting the American South (Project)

University of Michigan. Humanities Text Initiative (Project)

University of Indiana. Victorian Women's Writers Project (Project)

Table of Contents

Introduction

> "The vaunted "information superhighway" would hardly be worth traveling if the landscape were dominated by industrial parks, office buildings, and shopping malls. Thanks to the Text Encoding Initiative, there will be museums, libraries, theaters, and universities as well." ~Charles Goldfarb

Whether we have been trained as librarians or "information specialists," we are all in the business of presenting information in an well-organized and useful manner. Before the advent of the Internet as one of the leading disseminators of information, the librarian's job was relatively straightforward. He or she was in charge of indexing, information databases, and a well-educated and organized reference staff. However, in these days of universal access to a huge amount of information in the blink of an eye, the "information specialist" must deduce a way to bring as much information to the (very interested but naïve) public in the most efficient and timely possible way.

The most common markup language of the web is HTML. Originally meant for the simple design and layout of a page, this markup language has been forced to its limits by avid web developers, people desperate to push the envelope of what HTML has to offer. Unfortunately, no matter how far we push that envelope, HTML is not the answer. For describing screen layout, HTML is bearable but limited. For describing *the text*, HTML is unacceptably limited. It is built for publication – not for information retrieval, and definitely not for scholarly study, philological or otherwise (Humanities). Because it can denote the appearance of a web site, but cannot capture the substance of a document, it has long been considered by the information community as inappropriate for presenting complex and/or intellectually rich documents on the web.

Because of the simplicity of HTML, the digital library community could not use it to make their text available via the web. Instead, they chose to mark up their documents using SGML (the Standard Generalized Markup Language) practices. However, SGML also has two formidable

problems. First, it is an unduly complex meta-language, which takes a great deal of time and intellectual energy simply to learn and use. Second, its parsers also tend to be complex, large and hard to reproduce. Consequently, they tend not to be incorporated into other pieces of software like web browsers. However, until 1998 SGML was the best solution available. Despite the language's complexity and the fact that the documents could not be universally viewed over the web without expensive and large application programs (like Dynaweb),[1] many digital libraries and archives around the world use SGML to make their texts available electronically.

For the past six years, the World Wide Web Consortium (http://www.w3.org) has been working on a new mark-up language, which would have the searching power of SGML and the ease of use of HTML. That language is XML, the eXtensible Markup Language. With this new tool, digital librarians, among others, will be able to provide access to their documents in a way that was not possible with simple SGML. Because it is a simplified version of SGML, it is not as large and cumbersome, and XML parsers can be bundled with traditional web browsers. In the near future, online searchers will have the power to search multiple digital libraries for research materials easily and quickly.

With the development of XML DTDs, digital librarians who have been using SGML to mark up their texts now have the ability to use XML as well. However, for all of the documents that have already been encoded, a transformation needs to take place, which will make the SGML documents XML compliant. This project intends to facilitate that transformation, by looking at one of the most widespread SGML applications, the Text Encoding Initiative, how its DTD works, how the new XML DTD has been altered, and finally how well the existing transformative perl scripts work on some characteristic digital library texts.

Text Encoding Initiative

History

The Text Encoding Initiative (TEI) is an international project formed in 1987 to develop guidelines for the preparation and interchange of electronic texts for scholarly research, and also to fulfill a broad range of applications by the language industries. The TEI began with a planning

conference convened by the Association for Computers and the Humanities (ACH), gathering together experts in the field of electronic texts, representing professional societies, research centers, and text and data archives. (Thumbnail History)

The conference attendees agreed that it was necessary to create a common text encoding scheme for researchers to use in both creating electronic texts and exchanging existing documents among text and data archives. This new scheme would replace the existing system in which every text provider and every software developer had to create and support their own format, since existing schemes were typically informal constructs which supported the specific interests of their creators, but were not built for general use. (Thumbnail History)

After the initial planning conference, three sponsoring organizations, the Association for Computers and the Humanities (ACH), the Association for Computational Linguistics (ACL), and the Association for Literary and Linguistic Computing (ALLC) began developing an encoding system for use in the creation of electronic texts for research. The first public draft, referred to as "TEI P1" was published for public comment in June 1990. The sponsoring organizations immediately began making revisions following the release of the first public draft of the TEI, and in April 1992 through November 1993, the second public draft, called "TEI P2" was released. In the spring of 1993, all published chapters were revised one last time, they added some materials, and the development phase of the TEI ended with the publication of the first "official" version of the Guidelines, TEI P3 (not labeled a draft), in May 1994. Since that time, the TEI has concentrated on making the Guidelines more accessible to users and on preparing supplementary material like tutorials and introductions. (Thumbnail History)

Research is important not because of the words on a page, but the information that those words convey, independent of any specific physical form. As a consequence, the TEI is interested in both textual and non-textual resources in electronic form, either as elements of research databases or pieces of non-paper publications. The TEI wanted to be ready for the emerging technologies like the Internet, which would be able to bring together text, graphics and audio into a "seamless information-bearing vehicle." By providing a description of information which is

independent of media, the TEI scheme would make the construction and exploitation of multimedia technology possible. (What is TEI?)

The TEI project initially had two goals: they had to decide which textual features should be encoded, and how that encoding should be represented "for loss-free, platform-independent, interchange." (What is TEI?) Early on in the project, the TEI authors chose the Standard Generalized Markup Language (SGML; ISO 8879) as the most appropriate tool, initially for the practical reason that to create a comparable language would be a major research project in itself. Despite some complexities and inelegancies, SGML has proved adequate to the needs of researchers, and until 1998-1999, was still dominating the software industry. Having decided the *method* for encoding, the TEI was then able to focus its efforts on the *expression* of the set of textual features. (What is TEI?)

A very large number (over 400) of textual feature definitions, expressed as SGML elements and attributes, is the end-result of the TEI's work.[2] These elements are grouped into *tag sets* of various kinds, and constitute a modular scheme which can be configured to provide hardware-, software-, and application- independent support for the encoding of all kinds of text in all languages and of all times. (What is TEI?)

The Full TEI DTD and TEI-lite

The TEI is unique because it allows for many possible DTDs, which can be tailored to the needs of a particular application in a way that is difficult or impossible with other general purpose DTDs. With the TEI scheme the user can build a unique DTD which matches his or her needs, but must do it in a way that makes interchange of information possible. When creating a DTD, the document designer may be as lax or as strict as the he or she feels is necessary, but it is important to reach a middle ground between the ease of following rules and the intricacy of handling real texts. This is especially true when the rules relate to texts which already exist: the designer may not have any idea about the original purpose of some ancient text and will therefore find it very difficult to make decisions regarding structure and meaning. On the other hand, if a new text is being prepared for a specific application, like for entry into some kind of textual

database, the more specific the rules the better. Even in the case where an existing text is being marked up, and the document designer has no idea how it was originally intended to work, it is probably a good idea to define a restrictive set of rules related to one particular view or hypothesis about the text. The document designer should remember that when marking up texts, *every* document type definition is an interpretation of a text. No single DTD can possibly encompass the "absolute truth" about any text. (Gentle Introduction)

This interchange and malleability of the TEI scheme is referred to as the Chicago Pizza model. Pizzas all have a few ingredients in common (cheese and tomato sauce); in Chicago, they may have entirely different forms of pastry base, and the consumer is expected to make his or her own selection of toppings. Using SGML syntax this has been summarized as follows (Organization):

```
<!ENTITY % base    "(deepDish | thinCrust | stuffed)" >
<!ENTITY % topping "(sausage | mushroom | pepper | anchovy ...)">
<!ELEMENT pizza - - (%base, cheese & tomato, (%topping;)* )>
```

In a similar manner, the TEI document designer builds the TEI DTD by bringing together the core tag sets (which, like the cheese and tomato sauce are always present), one "base" tag set (like the changeable pastry base), and his or her own selection of "additional" tag sets (or toppings).

The term *tag set* represents a collection of SGML elements and attributes definitions. These tag sets are the basic organizing principles of the TEI scheme, and are divided into four groups (Organization):

- **core** tag sets refer to and define elements which are probably going to be needed by all documents, and are therefore available by default.
- **base** tag sets refer to and define particular classes of document where the overall structure may vary; generally only one base tag set is necessary for a given document.
- **additional** tag sets refer to and define sets of elements which may be found in any class of document but which are typically associated with some application or subject area.
- **auxiliary** tag sets comprising elements with highly specialized roles, typically for description of some part of the encoding scheme, and which make up a DTD independent of the main one.

Elements generally appear in only one tag set, though the TEI P3 allows for the redefinition of elements within different base tag sets. Elements may not be defined in more than one additional tag set (Organization).

The Core Tag Sets

Whenever the main TEI DTD is called, two core tag sets are *always* included. The tags and attributes in these two sets are therefore available to any TEI document(Structure). The parameter entities and the files they refer to, are:

TEI.core.dtd—refers to the file teicore2.dtd,

TEI.header.dtd—refers to the file teihdr2.dtd.

The Base Tag Sets

The base tag sets define the basic structure of different text types. The basic units of prose (`paragraph, section, chapter, etc`), for example, are not equal to those of drama (`act, scene, direction, soliloquy`).

Generally, one base tag set must be selected for any TEI-conformant document. If none, or more than one base tag set is selected, errors will occur because the same elements may be defined differently in different base tag sets. However, the TEI *has* allowed for documents which bring together structurally dissimilar elements or require elements from more than one base. In either of those cases, the document designer may use either the “mixed base” or the “general base” tag set. When these bases are called, the encoder must specify which of the other bases are to be combined (Structure).

To invoke the base tag set for prose, the encoder has to select a base tag set by identifying the appropriate SGML parameter entity as INCLUDE. For example:

```
<!ENTITY % TEI.prose 'INCLUDE' >
```

The entities used to select the different base tag sets, and the files containing the SGML declarations for each base, are listed below (Structure).

- **TEI.prose** – selects the base tag set for prose, contained in teipros2.dtd.
- **TEI.verse –** selects the base tag set for verse, contained in teivers2.dtd and teivers2.ent.

- **TEI.drama –** selects the base tag set for drama, contained in teidram2.dtd and teidram2.ent.
- **TEI.spoken –** selects the base tag set for transcriptions of spoken texts, contained in teispok2.dtd and teispok2.ent.
- **TEI.dictionaries** – selects the base tag set for print dictionaries, contained in teidict2.dtd and teidict2.ent.
- **TEI.terminology** – selects the base tag set for terminological data files, contained in teiterm2.dtd, teiterm2.ent, teite2n.dtd, and teite2f.ent.
- **TEI.general** – selects the generic mixed-mode base tag set, contained in teigen2.dtd.
- **TEI.mixed** – selects the base tag set for free mixed-mode texts, contained in teimix2.dtd.

As shown in the above list, each base tag set normally contains one or two system files: a required file (with the extension .dtd) which defines the elements in the tag set and their attributes, and an optional file (with the file extension .ent) which defines any global attributes or specialized element classes that tag set enables. (Structure)

The Additional Tag Sets

Encoders use the additional tag sets because they want to do different types of analysis and processing with their TEI application. They are optional, and are not necessary for the application to work. However, they do increase the potential and power of the application. Additional tag sets are compatible with each other and with every tag set and so can be used in any encoding project. They are similar to base tag sets, invoked by identifying the appropriate parameter entity as INCLUDE. The relevant parameter entities, and the files containing the additional tag sets, are these (list is from *Structure*):

- **TEI.linking** -- embeds the files teilink2.dtd and teilink2.ent, with tags for linking, segmentation, and alignment
- **TEI.analysis** -- embeds the files teiana2.dtd and teiana2.ent, with tags for simple analytic mechanisms
- **TEI.fs** -- embeds the file teifs2.dtd, with tags for feature structure analysis

- **TEI.certainty** -- embeds the file teicert2.dtd, with tags for indicating uncertainty and probability in the markup
- **TEI.transcr** -- embeds the files teitran2.dtd and teitran2.ent, with tags for manuscripts, analytic bibliography, and transcription of primary sources
- **TEI.textcrit** -- embeds the files teitc2.dtd and teitc2.ent, with tags for critical editions
- **TEI.names.dates** -- embeds the files teind2.dtd and teind2.ent, with specialized tags for names and dates
- **TEI.nets** -- embeds the file teinet2.dtd, with tags for graphs, digraphs, trees, and other networks (...not to be confused with the graphics markup of TEI.figures)
- **TEI.figures** -- embeds the files teifig2.dtd and teifig2.ent, with tags for graphics, figures, illustrations, tables, and formulae (...not to be confused with the graph-theoretic markup of TEI.nets)
- **TEI.corpus** -- embeds the file teicorp2.dtd, with tags for additional tags for language

Like the base tag sets, the additional tag sets normally contain one or two system files: a required file (with the extension .dtd) which defines the elements in the tag set and their attributes, and an optional file (with the file extension .ent) which defines any global attributes or specialized element classes that tag set enables. (Structure)

Figure 1 is an example of the start of a minimal TEI-conformant document in which the base tag set for drama has been selected together with the additional tag sets for linking and figures:

```
<!DOCTYPE tei.2 [
<!ENTITY % TEI.drama "INCLUDE">
<!ENTITY % TEI.linking "INCLUDE">
<!ENTITY % TEI.figures "INCLUDE">
]>
<tei.2>
<!-- content of document here -->
```

Figure 1: Base tag set for drama, additional tag sets for linking and figures

As the above discussion shows, It is possible for anyone looking at the document to tell which TEI tag sets are required because the selection has been stated explicitly by declarations within the DTD subset. If the document designer wants to make any amendments or changes to the TEI definitions like renaming elements or making new ones, all he or she has to do is define those elements in the DTD. (Gentle Introduction).

An Example DTD

A DTD is simply a set of declarative statements, using the syntax defined in the SGML standard. Figure 2 refers to a simple model of a poem.

```
<!ELEMENT anthology      - -  (poem+)>
<!ELEMENT poem           - -  (title?, stanza+)>
<!ELEMENT title          - O  (#PCDATA) >
<!ELEMENT stanza         - O  (line+)   >
<!ELEMENT line           O O  (#PCDATA) >
```

Figure 2: Simple poem DTD (from Gentle Introduction)

Figure 2 shows five typical SGML element declarations. Each declaration is enclosed by angle brackets; the opening bracket is followed by an exclamation mark, which is in turn followed immediately by a SGML-defined keyword which specifies the kind of object being declared. This keyword is followed by two characters indicating minimization rules, which is again followed by the content model and the closing bracket. Each of these parts is discussed further below. White space – that is one or more blanks, tabs or new lines – separates each part of the declaration (Gentle Introduction).

The first part of each declaration above gives the generic identifier of the element which is being declared, for example poem, title, etc. It is possible to declare several elements in one statement, as discussed below.

Minimization Rules

After the keyword and the generic identifier, the declaration identifies the minimization rules for the element being defined. These rules tell whether start- and end-tags must be present

in every occurrence of the element. They are expressed as two characters, divided by white space. The first character relates to the start-tag, and the second to the end-tag. The defined minimization characters are either a hyphen – indicating that the tag must be present – or a letter O (for ”omissible" or ”optional") meaning the tag may be dropped. So for figure 2, that means every element except <line> has to have a start-tag and only the <poem> and <anthology> elements must have end-tags (Gentle Introduction).

Content Model

The third part of each declaration, which is enclosed in parentheses, is called the “content model” of the element. It indicates what kind of information the encoder may legitimately include within each element. Contents are defined by either using special reserved words or by relating them to other elements. There are several reserved words, #PCDATA being by far the most common, as in figure 2. PCDATA is an abbreviation for *parsed character data*, and it simply means that the element being defined may contain any valid character data. If we think of an SGML declaration like a family tree, with a single ancestor at the top (in the case of figure 2, that would be <anthology>), then if we were to follow the branches of the tree downward (from <anthology> to <poem> to <stanza> to <line> and <title>) we will eventually come to #PCDATA. In figure 2, <title> and <line> are defined with #PCDATA (Gentle Introduction).

Occurrence Indicators

In figure 2, the declaration for <stanza> says that a stanza must consist of one or more lines. It communicates this information by the use of an occurrence indicator (the plus + sign), which specifies the number of times an element may occur. There are three occurrence indicators in SGML: the plus sign (+) – meaning that there may be one or more occurrences of the element, the question mark (?) – meaning there may be at most one and perhaps no occurrence, and the asterisk or star (*) – meaning the element may either be absent or appear one or more times. So if the content model for <stanza> were (LINE*), it would be possible to have a stanza that had no lines, as well as stanzas that had one or more lines. Comparatively, if the content model for <stanza> were (LINE?), empty stanzas again would be allowed, but any stanza could not have more than a single line (Gentle Introduction).

Model Groups

In figure 2, the components of each content model are either single elements or #PCDATA. It is also possible to define content models in which the components are lists of elements, combined by group connectors. To demonstrate this, we'll expand the example to include non-stanzaic types of verse. For this example, poems will be defined as stanzaic, couplet or blank. A blank-verse poem is made up simply of lines. A couplet is classified as a <line1> followed by a <line2> (Gentle Introduction).

```
<!ELEMENT couplet O O (line1, line2) >
```

The elements <line1> and <line2> have the same content model as the existing <line> element from figure 2, and can share the same declaration. When this situation arises, it is useful to define a name group as the first component of a single element declaration, rather than give a series of declarations differing only in the names used. A name group is a list connected by any group connector and enclosed in parentheses:

```
<!ELEMENT (line | line1 | line2) O O (#PCDATA) >
```

The declaration for the <poem> element can now be changed to include all three possibilities:

```
<!ELEMENT poem - O (title?, (stanza+ | couplet+ | line+) ) >
```

Which means, a poem consists of an optional title (?), followed by one or several stanzas(+), **or** one or several couplets (+), **or** one or several lines (+). Notice difference between this definition and the following:

```
<!ELEMENT poem - O (title?, (stanza | couplet | line)+ ) >
```

The second version, by applying the occurrence indicator to the group rather than to each element within it, would allow a **single** poem to contain a mixture of stanzas, couplets or blank verse (Gentle Introduction).

In this way it is possible to build up complex models, which in turn matches the structural complexity of many types of text. For another example, consider the case of stanzaic verse with a refrain or chorus appears. A refrain can be repetitions of the line element, or it may simply be text, not divided into verse lines. A refrain can appear at the start of a poem only, or as an

optional addition following each stanza. This could be expressed by a content model such as the following:

```
<!ELEMENT refrain - - (#PCDATA | line+)>

<!ELEMENT poem    - O (title?,

      ( (line+)

      | (refrain?, (stanza, refrain?)+ ) )) >
```

Figure 3: Stanzaic verse with refrain (from Gentle Introduction)

A poem consists of an optional title (?), followed **either** by a series of lines, or an un-named group, which begins with an optional refrain (?), followed by one of more occurrences (+) of another un-named group (), each member of which is composed of a stanza followed by an optional refrain (?). The sequence “refrain - stanza - stanza – refrain” follows this pattern, as does the sequence “stanza - refrain - stanza - refrain.” However, “refrain - refrain - stanza – stanza” does not does not follow the model, and neither does the sequence “stanza - refrain - refrain - stanza." (Gentle Introduction).

At its most simple, the document type definition is made up simply of at least one base, and possibly more, document type definitions. For example:

```
<!DOCTYPE my.dtd [
<!-- all declarations for
MY.DTD go here -->
...
]>
<my.dtd>
This is an instance of a
MY.DTD type document
</my.dtd>
```

Figure 4: simple DTD (from Gentle Introduction)

More common, the document type definition will be stored in a separate file referred to in quotes, as shown in Figure 5:

```
<!DOCTYPE tei.2 system
"tei2.dtd" [
]>
<tei.2>
This is an instance of an
unmodified TEI type
document
</tei.2>
```

Figure 5: DTD in separate file (from Gentle Introduction)

The Document Instance

The document instance is the content of the document. It contains only text, markup and general entity references, and does not contain any new declarations. An easy way to build up large documents in a modular fashion might be to use the DTD to declare entities for the individual pieces or modules, thus (Gentle Introduction):

```
<!DOCTYPE tei.2 [
      <!ENTITY chap1 system "chap1.txt">
      <!ENTITY chap2 system "chap2.txt">
      <!ENTITY chap3 "-- not yet written --">
      ]>
<tei.2>
      <teiHeader> ... </teiHeader>
      <text>
      <front> ... </front>
            <body>
                  &chap1;
                  &chap2;
                  &chap3;
      ...
            </body>
      </text>
```

Figure 6: Modular buildup of large documents (from Gentle Introduction)

In figure 6, the DTD contained in file `tei2.dtd` has been expanded by entity declarations for each chapter of the document. The first two declarations refer to the file which contains the text of those particular chapters; the third declaration is a dummy reference, indicating that the text is “not yet written.” In the document instance, the entity references `&chap1;` etc.. will be fetched by the parser to give the requisite contents. Again, the chapter

files themselves will not contain any element, attribute list, or entity declarations—just tagged text (Gentle Introduction).

TEI-lite

For many applications, even relatively complex ones, the full TEI DTD is unwieldy and unnecessarily intricate. Therefore, a more manageable DTD was developed, called TEI-Lite,[3] which is concerned more with structure than content. In selecting from the several hundred SGML elements defined by the full TEI scheme, the developers of TEI-lite tried to identify a useful "starter set," made up of the elements about which almost every user should know.

Their goals in defining this subset may be summarized as follows (TEI-Lite Introduction):

- it should include most of the TEI "core" tag set, since this contains elements relevant to virtually all text types and all kinds of text-processing work;[4]
- it should be able to handle a wide variety of texts;
- it should be useful for the production of new documents as well as encoding existing ones;
- it should be usable with a wide range of existing SGML software;
- it should be derivable from the full TEI DTD described in the TEI Guidelines;
- it should be as small and simple as is consistent with the other goals.

Nuts and Bolts

All TEI-conformant texts contain (1) a *TEI header*, tagged as a <*teiHeader*> element; and (2) the transcription of the text proper, tagged as a <*text*> element.

The TEI header provides information comparable to that provided by the title page of a printed text. It has four parts: a bibliographic description of the machine-readable text, a description of the way it has been encoded, a non-bibliographic description of the text – a *text profile*, and a revision history (TEI-Lite Introduction).

A TEI text may be *unitary* (a single work) or *composite* (a collection of single works, such as an anthology). In either case, the text may have an optional *front* or *back*. In between is the

body of the text, which, in the case of a composite text, may consist of *group*s, each containing more groups or texts (TEI-Lite Introduction).

A unitary text will be encoded using an overall structure like the one in figure 7:

```
<TEI.2>
    <teiHeader> [ TEI Header information ]
</teiHeader>
    <text>
        <front> [ front matter ... ]   </front>
        <body>  [ body of text ... ]   </body>
        <back>  [ back matter ...  ]   </back>
    </text>
```

Figure 7: Unitary text (from TEI-Lite Introduction)

A composite text also has an optional front and back. In between occur one or more groups of texts, each with its own optional front and back matter. A composite text will be encoded using an overall structure like the one in figure 8:

```
<TEI.2>
    <teiHeader> [ header information for the composite ] </teiHeader>
    <text>
        <front> [ front matter for the composite  ]      </front>
        <group>
           <text>
              <front> [ front matter of first text ] </front>
              <body>  [ body of first text  ]          </body>
              <back>  [ back matter of first text ]    </back>
           </text>
           <text>
              <front> [ front matter of second text]  </front>
              <body>  [ body of second text  ]          </body>
              <back>  [ back matter of second text ]    </back>
           </text>
           [ more texts or groups of texts here ]
        </group>
        <back>      [ back matter for the composite  ]      </back>
    </text>
</TEI.2>
```

Figure 8: Composite text (from TEI-Lite Introduction)

Encoding the Body

A simple TEI document at the textual level consists of the following elements (TEI Introduction):

<front>
contains any preface material (headers, title page, prefaces, dedications, etc.) found before the start of the text proper.
<group>
contains unitary texts or groups of texts.
<body>
contains an entire single unitary text, excluding any front or back matter.
<back>
contains any appendixes, etc., following the main part of a text.

The SGML TEI DTD has been an adequate tool for digital librarians wishing to make their text available electronically. However, even with all of the effort expended on tagging the text appropriately, these texts are still not reaching wide audiences in a easily useful way. That is to say that they are highly functional to people with SGML processors, and available as HTML text to everyone else. With the advent of XML, the eXtensible Markup Language, all of the digital librarian's work of marking up and tagging text would be fully functionally available to a vast audience of people via the web in a searchable and interactive form.

What is XML?

In 1996 the W3 Consortium (http://www.w3.org), an independent standards organization comprised of members of industry, government and academia, began working on an application to enable generic SGML on the Web (Bray). In February 1998, they released a recommendation for XML, the eXtensible Markup Language (W3C, 1998). XML is not a single tag set like HTML, but is a way to *define* tag sets formally, with a clear definition of conformance. XML is a meta-language, not a stand-alone application, which allows other languages to be formally defined, and is the foundation upon which a wide variety of applications dealing with structured data can be built (W3C, 1998). The goals of XML were thus: it should be usable on the web, but also functional on a wide variety of applications. It should be compatible with SGML, easy to process, have few optional features (ideally none), be human-legible, reasonably clear, finished quickly, formally and concisely specified, and the documents should be easy to create (Bray). With these goals, the designers of XML were hoping to rid their new meta-language of the two major problems that had plagued SGML – namely the complexity of the language itself, and its incompatibility with the web-based applications. As a subset of SGML, XML is very similar in form

and function. The major differences reside in XML's simplicity and ease of use, its size and web availability.

XML allows the author to mark up text documents with a **self-defined** tag set and structure. Like its predecessor SGML, appropriately named tags give semantic meaning to the encoded content, and hierarchical combinations of tags give structure to the data. With the advent of XML, the web community (as opposed to the SGML community) now has the ability to mark up its texts with regard to content *as well as* layout, in a language that – hopefully once the newest generation of browsers are released – can be universally accessed over the Internet. Information specialists will be able to present *information* on the World Wide Web rather than simply providing good-looking sites. XML allows for smarter pages and database exchanges, which will make the search for information easier and vastly more efficient than we know today. The average web user will have a much more powerful tool, once XML becomes the standard language of the web.

Components of XML Documents

Every SGML (and thus XML) based document has both a logical and a physical structure. An XML document is divided into two parts: the prolog and the "*root*" element or document entity. Just as in SGML, the XML document is composed of declarations, elements, comments, character references, and processing instructions, all of which are indicated in the document by explicit markup.

Prolog

Unlike any general SGML document, in a well-formed XML document the prolog and all of its components *is optional*. However, the W3C strongly recommends that all XML documents should begin with an *XML declaration* which specifies the version of XML being used:

```
<?xml version="1.0" encoding="UCS2" standalone="yes">
```

"xml version=1.0" refers to the version of xml being used, "encoding=UCS2" refers to the character set, and "standalone=yes" refers to the fact that this example document was authored either with no document type definition, or with an internal document type definition.

and may be rejected by the processing application.[6]

Just like SGML, The DTD may either live inside document type declaration or external to the document proper. If the document is using an already defined document type definition, the document type declaration will refer to an external file containing the relevant markup declarations (the external subset), e.g.:

```
<!DOCTYPE poems SYSTEM "http://www.poems/dtds/oldpoem.dtd">
```

If the DTD is defined internally, it will be within square brackets, e.g.:

```
<!DOCTYPE poems [
<!ELEMENT oldpoem (element1, element2...) >
<!ELEMENT ...(#PCDATA) >
<!ELEMENT ...(#PCDATA) >
]>
```

Figure 9: Internally defined DTD

Some elements do not require any contents as such. They are simply placeholders that indicate where a certain process is to take place. A special form of tag is used in XML to indicate empty elements that do not have any contents, and therefore have no end-tag. For example, a `<graphic/>` element is typically an empty element that acts as a placeholder for the graphical

part of a figure while an optional <caption> element identifies any text associated with the illustration.

```
<!ELEMENT poems (element1, element2,
element3,(para|figure)? >

<...>

<!ELEMENT figure  (graphic, caption?) >

<!ELEMENT graphic EMPTY >

<!ELEMENT caption (#PCDATA) >
```

Figure 10: Allowing Empty Tags

The element declarations in figure 10 show how to expand the model for an <oldpoem> to allow it to include figures as well as text. The element notes show that the <graphic> and optional <caption> together make up a <figure>, which would typically be placed at the same level as a text paragraph.

Additional Components

In addition to its extensibility, and document type definition, CDATA and Entity Reference give extra power to XML documents.

Entity and Character References

One of the greatest characteristics of SGML and XML is that an author can easily add standard text to a file, and can also handle characters that are outside the standard character set, but which are available on certain output devices. Commonly used text can be declared within the DTD as a text entity. A typical text entity definition could take the form:

```
<!ENTITY Mac "the Scottish Play" >
```

Once such a declaration has been made in the DTD users can use an entity reference of the form &Mac; in place of the play's full name. In the output, "the Scottish Play" will appear. One great advantage of using this technique is that, should the quote "the Scottish Play" referred to by

the mnemonic change, only the entry in the DTD needs to be changed, because the entity reference will automatically call in the current definition.

Text stored in another file it can also be incorporated into a file using entity references. In this case the entity declaration in the DTD identifies the location of the file containing the text to be referenced, e.g.:

```
<!ENTITY appendix SYSTEM "http://www.mypoem.com/pub/appendix.xml">
```

and the entity reference `&appendix;` shows where the file is to be added to the main text stream.

CDATA

In some cases, text may contain numerous instances of the delimiter characters, but not contain any actual markup. It would be unrealistic and/or detrimental to convert all instances of those characters into entity references. XML allows one to store text as character data which by definition may not contain other markup. Such a section of character data is known as CDATA. The beginning and end of CDATA sections are delimited by <![CDATA[and]]> respectively.

For example, if a document requires non-standard characters, special system-dependent entities can be declared to show how the characters can be generated. A typical entry might read:

```
<!ENTITY Ouml    "&#214;">
```

When the string Ö is encountered in the text the computer will replace it by the code whose decimal value is 214. Alternatively the decimal character number, or its hexadecimal equivalent, preceded by x, can be used directly as part of a character reference, e.g. &#xO7 is used to generate *Ö*.

Happily, SGML requires DTDs in order to work at all, while XML only requires them if the author wants to have a valid document. The most obvious reason for wanting a valid document is that it would assure interoperability. If all of the digital libraries in the world conform to one specific DTD, then searching and using their databases would be easy, and the information contained in the digital library would become vastly more authoritative. If, on the other hand, each digital

library were to develop its own XML DTD, the information would be disparate and sparse – there wouldn't be any interoperability and researchers would have the same difficulties finding pertinent information that they have now – for example, they would still have to do multiple searches on the same subject, or if they got a null return, they might not be sure if some piece of information really wasn't available or if it simply wasn't marked up. For the same reasons that the TEI developed their SGML TEI-DTD, they realized that they had to develop an XML version. They wanted to make sure that all of the effort and energy involved in making a digital library available to the public generated the most useful information in the least wasteful manner. The TEI founders know that if the web community wants XML to realize its potential as a powerful searching and indexing tool, those SGML version DTDs must be transformed XML version DTDs, and the documents' mark up must also be transformed from SGML to XML.

Using XML in the TEI

In July of 1999, C.M. Sperberg-McQueen developed an XML version of the TEI-Lite DTD. This was an important breakthrough, allowing the digital library community to develop XML documents on the web. In this section I will briefly explain the XML TEI-Lite DTD, but if the whole process is interesting to you, please go to *Construction of an XML Version of the TEI DTD* (at http://www.uic.edu/orgs/tei/ed/edw69.html). This is an unpublished document, is not yet a formal publication, and can not be quoted in published material. However, I will briefly explain Sperberg-McQueen's process.

As stated above, XML defines a syntax for document type definitions similar to that provided by SGML, but is somewhat more restrictive. Specifically, XML allows neither inclusion nor exclusion exceptions, and prohibits the ampersand connector.

Modifying an existing SGML document type definition (DTD), such as the TEI DTD, to conform to XML entails (Construction):

- removing tag omissibility information
- normalizing references to parameter entities by making sure that they always end with a semicolon

- removing ampersand ”&” connectors
- normalizing mixed-content models to the canonical form prescribed by XML (#PCDATA must come first, the list of sub-elements must be flat, and the occurrence indicator must be a star)
- removing exclusion exceptions
- removing inclusion exceptions

1. Removing Tag Omissibility Information

Removing tag omissibility was accomplished by a simple editor script. The strings - -, - O, O -, and O O are legal in a DTD only as tag omissibility information, within comments, or within literals. In the TEI DTDs, they do not occur within literals (Construction).

2. Normalizing parameter-entity references

In the short term, parameter-entity references were normalized using the script mentioned above.

In the long run, all content models in the tagdocs of TEI P3 will be systematically normalized by adding semicolons to parameter-entity references which currently do not have them (Construction).

3. Ampersand connectors

To remove the ampersand connectors either the content model must be rewritten as a set of alternative sequences or it must be revised entirely. In the case of the TEI, most uses of the ampersand connector were design errors, so the content models will simply be revised.

The following content models use & in TEI P3 (Construction):

- <cit> (part of the core)
- <respStmt> (part of the core)
- <publicationStmt> (part of the header)
- <graph> (part of the additional tag set for networks and graphs)

4. Normalizing mixed-content models

The following elements use the keyword #PCDATA in ways that must be changed to be legal in XML (Construction):

- <sense> (dictionaries)
- <re> (dictionaries)
- <persName> (names and dates)
- <placeName> (names and dates)
- <geogName> (names and dates)
- <dateStruct> (names and dates)
- <timeStruct> (names and dates)
- <dateline> (default text structure)

5. Exceptions

Removing inclusion and exclusion exceptions involved changing the set of documents accepted by the DTD (Construction).

6. Exclusions

To rewrite declarations without exclusion exceptions Sperberg-McQueen simply removed the exception, and added an application-specific constraint which must be checked outside the SGML parser which says that the excluded element type can not occur within the element type which excluded it (pretty obviously).

For example, the TEI <s> element (for end-to-end segmentation on the level of the orthographic sentence) is currently declared:

```
<!ELEMENT s  - -  (%phrase.seq)  -(s) >
```

An XML-compatible TEI DTD would replace this with:

```
<!ELEMENT s %phrase.seq;  >
<!--* CONSTRAINT:  <s> must not occur within
    * an <s>, i.e. Ancestor(1,s) = NIL
    *-->
```

The important change is the removal of the exclusion exception. In addition, the tag omissibility indicators and the parentheses around `phrase.seq` were removed (Construction).

7. Inclusions

Removing inclusion exceptions required reproducing their effect in the content model of each element type which can occur as a descendant of the element type bearing the inclusions.

Inclusions make included elements legal at any location in a content model, without changing the requirements of the basic content model.

As of September 1994, the following elements have inclusion exceptions in TEI P3 (Construction):

- <entry> (includes <anchor>)
- <entryFree> (includes %m.dictionaryParts; | %m.phrase; | %m.inter;)
- <eg> (includes %m.dictionaryParts; | %m.formPointers;)
- <orgName> (includes <orgtitle>, <orgtype>, and <orgdivn>)
- <text> (includes %m.globincl;, i.e. <alt>, <altGrp>, <cb>, <certainty>, <fLib>, <fs>, <fsLib>, <fvLib>, <index>, <interp>, <interpGrp>, <join>, <joinGrp>, <lb>, <link>,
- <linkGrp>, <milestone>, <pb>, <respons>, , <spanGrp>, and <timeline>)
- <lem> (includes %m.fragmentary;, i.e. <lacunaEnd>, <lacunaStart>, <witEnd>, and <witStart>)
- <rdg> (includes %m.fragmentary;)
- <termEntry> (the version in the nested DTD includes %m.terminologyInclusions;, i.e. <date>, <dateStruct>, <note>, <ptr>, <ref>, <xptr>, and <xref>)

8. Elements Requiring Manual Intervention

There are a number of elements which required manual intervention. This means that their changes were not universal, nor were they regular. These elements occurred in most tag-sets: core, basic text-structure, front matter, header, verse, drama, spoken text, dictionary, terminology, segmentation and alignment, analysis and interpretation, feature structures, names and dates, text-criticism, graphs and digraphs, and tables.

9. The problem of the dictionary chapter

The dictionary chapter proved to be a major problem in the transition from SGML to XML. However, because this paper does not deal with dictionaries, and the project I decided to work on involves no dictionaries, I decided not to include a complete discussion of the changes. However, as with the rest of this section, if this transformation is interesting, please go to C.M. Sperberg-McQueen's *Construction of an XML Version of the TEI DTD*.

The XML Translation Experiment

This experiment is concerned primarily with translating some characteristic SGML texts into XML compliant documents. With Sperberg-McQueen's development of the TEI XML DTD, digital librarians who have been using SGML to mark up their texts now have the ability to use XML as well. This is wonderful news for librarians who want to make their work available to larger numbers of people, and texts that will be encoded in the near future will certainly be tagged in an XML compliant manner. However, for all of the documents which have already been encoded, a transformation needs to take place which will make the SGML documents XML compliant. This project intends to facilitate that transformation, by testing two existing transformative scripts on some SGML documents from three digital libraries.

I ran each script on each text, and reported the output. With this information, the person who writes the next version of the script will have a better idea of what needs to be fixed, and how to fix it. Once this script has been perfected, it will be a boon for digital libraries around the world. They will be able to provide access to their documents via the web, to ever increasing numbers of people.

I used texts from three different sources: the University of North Carolina's *Documenting the American South* (http://www.metalab.unc.edu/docsouth), the University of Michigan's *Humanities Text Initiative* (http://www.hti.umich.edu/), and the University of Indiana's *Victorian Women's Writers Project* (http://www.indiana.edu/~letrs/vwwp). *Documenting the American South* and the *Victorian Women's Writers Project* provided me with texts they thought would be typical and/or problematic, and I randomly chose texts from the *Humanities Text Initiative*.

Documenting the American South provides access to digitized primary materials that offer Southern perspectives on American history and culture. Presently, *Documenting the American South* consists of five projects: slave narratives, first-person narratives, Southern literature, Confederate imprints, and materials related to the church in the black community. (DocSouth). The titles from *Documenting the American South* are given in Table 1:

Author	Abbreviation	Title	Publisher & Date
Benjamin Griffith Brawley 1882-1939	brawley	*Women of Achievement: Written for The Fireside Schools* , *Under the Auspices of the Woman's American Baptist Home Mission Society*.	[Chicago, Ill.]: Woman's American Baptist Home Mission Society, c1919.
William Wells Brown, ca. 1814-1884	brown	*The Black Man: His Antecedents, His Genius, and His Achievements.*	New York: T. Hamilton; Boston: R. F. Wallcut, 1863
(--no author--)	alabama	*Ordinances and Constitution of the State of Alabama: with the Constitution of the Provisional Government and of the Confederate States of America*	Montgomery: Barrett, Wimbish, 1861
Amanda Smith, 1837-1915	smith	*An Autobiography. The Story of the Lord's Dealings with Mrs. Amanda Smith the Colored Evangelist; Containing an Account of Her Life Work of Faith, and Her Travels in America, England, Ireland, Scotland, India, and Africa, as an Independent Missionary*.	Chicago: Meyer & Brother, 1893
Protestant Episcopal Church in the Confederate States	catechsl	*A Catechism to Be Taught Orally To Those Who Cannot Read; Designed Especially for the Instruction of the Slaves*	Raleigh: Office of "The Church Intelligencer," 1862

Table 1: Texts from *Documenting the American South*

The Humanities Text Initiative (HTI) is an electronic archive of volumes of American poetry, mostly from the 19th century, although it also includes some 18th and 20th century texts. The texts I used from this collection are given in Table 2:

Author	Abbreviation	Title	Publisher & Date
A. Bronson Alcott	alco	*Ralph Waldo Emerson : an estimate of his character and genius : in prose and verse*	Boston, A. Williams & Co. 1882
William Stanley Braithwaite	brait	*Anthology of magazine verse for 1920 and year book of american poetry*	Boston, Small, Maynard & Company, 1920
Madison Cawein.	cawe	*Ode read August 15, 1907, at the dedication of the monument erected at Gloucester, Massachusetts, in commemoration of the founding of the Massachusetts Bay colony in the year sixteen hundred and twenty-three*	Louisville, KY, John P. Morton & Company, Incorporated, 1908
Richard Watson Gilder	gilde	*A Book of Music*.	New York: A Century Company, 1906.
Sidney Lanier.	lanie	*Poem Outlines*.	New York: Charles Scribner's Sons, 1908
Henry Wordsworth Longfellow.	long	*Song of Hiawatha*.	Boston: Ticknor and Fields, 1856.
Edna St. Vincent Millay.	milla	*Second April.*	New York: Harper & Brothers, 1921.

Table 2: Texts from the *Humanities Text Initiative*

Finally, I got some texts from The *Victorian Women Writers Project* at the University of Indiana. The goal of the *Victorian Women Writers Project* is to produce highly accurate transcriptions of works by British women writers of the 19th century, which includes anthologies, novels, political pamphlets, religious tracts, children's books, and volumes of poetry and verse drama. Table 3 shows the texts from the *Victorian Women Writers Project*.

Author	Abbreviation	Title	Publisher & Date
Marie Corelli.	corelli	*The Treasure of Heaven.*	London: Archibald Constable, 1906
Sarah Ellis.	ellis	*Sons of the Soil*.	London: Fisher, Son & Co., [1840].
Mrs Humphry Ward.	ward	*Story of Bessie Cottrell*.	London: Smith, Elder, 1895.

Table 3: Texts from the *Victorian Women's Writers Project*

I used two scripts to translate the texts from SGML to XML. The first is a perl script written by Jean Daniel Fekete (e-mail: Jean-Daniel.Fekete@emn.fr), and the second is an addendum to the first, written by Charles Viles (at http://ils.unc.edu/~wingm/tei2xtei.pl) .

The process I followed:

1. downloaded the files from various institutions.
2. ran the original and updated scripts.
3. Edited each xml document that resulted from each script
    a. Removed <!SGML lines and entity references at the beginning.
    b. Changed DOCTYPE to reference appropriate DTD <!DOCTYPE TEI.2 PUBLIC "-//TEI//DTD TEI Lite XML ver. 1.3//EN" "http://www.uic.edu/orgs/tei/lite/teixlite.dtd"
    c. Removed [?STYLESPEC and <?NAVIGATOR lines from DOCTYPE element
    d. Changed <!ENTITY of all images from "JPEG" to "jpeg" (or from "GIF" to
    e. Added entities for copy, ldquo, rdquo, lsquo, rsquo, mdash
    f. Added ISO references (as per Viles and BonHomme Instructions)

4. used a stylesheet written by Viles for a separate project.
5. checked if the documents could be viewed using Internet Explorer.
6. checked the documents with Richard Tobin's XML Well-Formedness Checker and Validator (at http://www.ltg.ed.ac.uk/%7Erichard/xml-check.html).

Criteria

There were three major criteria for my tests. First, I was interested in whether or not the document could be viewed with the one current browser that has XML capabilities, Internet Explorer. Initially, this seemed like the most obvious test to run, because all you have to do is see if the document shows up when you load it. However, after working on the project for a few hours, I decided that the Internet Explorer option, while interesting, was a little too simplistic. For example, most of the documents which did "show up" on IE only showed up partially. Usually the first section was viewable, but the body was not. This is related, I think, to the fact that the browser requires a stylesheet to view XML documents, and the stylesheet I used was not a general one. I worried that the documents were not showing up simply because they did not fit into the existing stylesheet. The second and third criteria are related to whether the output documents are well-formed and valid. By using Richard Tobin's *XML Well-Formdness Checker and Validator*, I could find out if the document was well-formed. If it was well-formed, I wanted to know if it was valid.

After I ran those tests and knew which documents were problematic, I divided the problems into four groups: Those documents which had problems with start/end tags, those which had unidentified entities, those which had undeclared elements, and those which had problems with the Ampersand "&" Connector and the ensuing white space.

Results

I broke the results into three groups: documents that were only well-formed, those which were well-formed and valid, and those that were not well-formed.

Title	Well-Formed?
alabama1(DS)	Y
ellis1 (VWW)	Y
brawley1 (DS)	Y
gild1 (HTI)	Y
catechsl1 (DS)	Y
ward1 (VWW)	N (adot entity)
smith1 (DS)	N (content model problems, oelig entity)
corelli1 (VWW)	N (doc ends too soon? Unidentified entity? Illegality problems?)
long1 (HTI)	N (expected </change> got </revisionDesc> @ 145)
gild-orig (HTI)	N (expected </LB>, got </ITEM> at line 157)
brait1 (HTI)	N (expected name but got <space> for entity @136)
cawe1 (HTI)	N (expected name but got <space> for entity @164)
milla1 (HTI)	N (hellip entity)
lanie1 (HTI)	N (mismatched end-tag, expected </foreign>, got </l>)
alco1 (HTI)	N (OHsagr entity)
denali…ames-try	N (PCDATA not allowed in unnamed entity, @115)
alabama-orig (DS)	N (UNDECLARED ELEMENTS)
alco-orig (HTI)	N (UNDECLARED ELEMENTS)
brait-orig (HTI)	N (UNDECLARED ELEMENTS)
brawley-orig (DS)	N (UNDECLARED ELEMENTS)
brown-orig (DS)	N (UNDECLARED ELEMENTS)
catechsl-orig (DS)	N (UNDECLARED ELEMENTS)
cawe-orig (HTI)	N (UNDECLARED ELEMENTS)
corelli-orig (VWW)	N (UNDECLARED ELEMENTS)
ellis-orig (VWW)	N (UNDECLARED ELEMENTS)
lanie-orig (HTI)	N (UNDECLARED ELEMENTS)
long-orig (HTI)	N (UNDECLARED ELEMENTS)
milla-orig (HTI)	N (UNDECLARED ELEMENTS)
smith-orig (DS)	N (UNDECLARED ELEMENTS)
ward-orig (VWW)	N (UNDECLARED ELEMENTS)
brown1 (DS)	N (unidentified oelig entity)

Table 4: Results of Well-Formedness Test

The original script (denoted by the filename-orig) was unsuitable, returning all unreadable documents. The second script was better, with a 31% success rate for outputting well-formed documents. The problems with those documents which were not well formed were: unidentified entities, start/end tags, and content model problems.

Unidentified entities is probably the easiest problem to fix – it's simply a matter of making sure that all of the entities defined in the original SGML text are accounted for in the XML text.

The nature of the start/end tags problem is a little more complex. The major difficulty is that the parser was expecting one end tag and got another. For example, in long1 (Henry Wordsworth Longfellow, *Song of Hiawatha* from the *Humanities Text Intitiative*), the system expected </change> and got </revisionDesc>. The document is marked up thus:

```
<revisionDesc><change><date>September
1996</date><respStmt><resp>UM DLPS</resp><name>Bill
Kowalski</name></respStmt><item>added table of
contents</item></revisionDesc>
```

The problem lies in the fact that in SGML the tags do not have to be closed. In XML, every tag *must* have an end tag. A third generation script would need to check and insert ending tags. This is a complicated procedure, because the script would have to refer to the DTD to see where to insert the ending tag.

The third major issue, the "content model" problem, only occurred in one document, smith1 (Amanda Smith, *An Autobiography. The Story of the Lord's Dealings with Mrs. Amanda Smith the Colored Evangelist…*from *Documenting the American South)*. The error message reads:

```
Warning: Content model for argument does not allow element p here
in unnamed entity at line 1116 char 13
```

I've looked at the SGML and the XML version of the document, and can not tell what the parser is referring to. At first I thought it might be the <p> tag, but those occur way before line 1116, and they also occur more often than once every 30 or so lines. Furthermore, every one of these statements (which go on for 2 pages) say that the error occurs at `char 13`. In many cases I can't find any "p" of any sort on the lines indicated.

The final problem, which again is somewhat mysterious, and I'm not qu

problem, is with the document corelli1 (Marie Corelli, *The Treasure of Heaven* from the *Victorian Women's Writers Project*):

- `Illegal UTF-8 start byte <0xa3> at file offset 151786 I/O error on stream <http://ils.unc.edu/~wingm/project/corelli1.xml>, ignore`

further errors

- Illegal UTF-8 start byte <0xa3> at file offset 151786 I/O error on stream <http://ils.unc.edu/~wingm/project/corelli1.xml>, ignore further errors

- Error: Document ends too soon in unnamed entity at line 2476 char 75

I assumed that the first two errors had something to do with the parser itself, but the final error, that the document ends too soon, is not correct. With further entries into the Validator, I am still receiving the same error messages. Perhaps the script corrupted this file.

Title	Well-Formed? Valid?
alabama1	Y (not valid: attribute of element note)
ellis1	Y (not valid: attribute of element one, allowed values)
brawley1	Y (not valid: unidentified entitites)
gild1	**Y (valid?)**
catechsl1	**Y (valid?)**
ward1	N (adot entity)
smith1	N (content model problems, oelig entity)
corelli1	N (doc ends too soon? Unidentified entity? Illegality problems?)
long1	N (expected </change> got </revisionDesc> @ 145)
gild-orig	N (expected </LB>, got </ITEM> at line 157)
brait1	N (expected name but got <space> for entity @136)
cawe1	N (expected name but got <space> for entity @164)
milla1	N (hellip entity)
lanie1	N (mismatched end-tag, expected </foreign>, got </l>)
alco1	N (OHsagr entity)
denali…ames-try	N (PCDATA not allowed in unnamed entity, @115)
alabama-orig	N (UNDECLARED ELEMENTS)
alco-orig	N (UNDECLARED ELEMENTS)
brait-orig	N (UNDECLARED ELEMENTS)
brawley-orig	N (UNDECLARED ELEMENTS)
brown-orig	N (UNDECLARED ELEMENTS)
catechsl-orig	N (UNDECLARED ELEMENTS)
cawe-orig	N (UNDECLARED ELEMENTS)
corelli-orig	N (UNDECLARED ELEMENTS)
ellis-orig	N (UNDECLARED ELEMENTS)
lanie-orig	N (UNDECLARED ELEMENTS)
long-orig	N (UNDECLARED ELEMENTS)
milla-orig	N (UNDECLARED ELEMENTS)
smith-orig	N (UNDECLARED ELEMENTS)
ward-orig	N (UNDECLARED ELEMENTS)
brown1	N (unidentified oelig entity)

Table 5: Results of Well-Formed, Validity Test

Of the five documents that are well-formed, two appeared to be valid as well. Those two texts are: gild1 (Richard Watson Gilder. *A Book of Music*. from the *Humanities Text Initiative*) and catechsl1 (Protestant Episcopal Church in the Confederate States. *A Catechism to Be Taught Orally To Those Who Cannot Read; Designed Especially for the Instruction of the Slaves*. from *Documenting the American South*). The other three documents which were well-formed but not valid had the following problems: unidentified entities, and allowed values in attribute element notes.

Alabama1 (*Ordinances and Constitution of the State of Alabama: with the Constitution of the Provisional Government and of the Confederate States of America*, from *Documenting the American South*) had the following error message (which goes on for 5 pages, with different line numbers):

```
Warning: In the attribute anchored of element note, NO is not one
of the allowed values in unnamed entity at line 3500 char 30
```

And at line 3500,

```
<note rend="sc" anchored="NO" place="margin">Political power in
the people.</note>
```

I randomly checked 7 more lines and each was this “note re

A related problem was also present in ellis1 (Sarah Ellis. *Sons of the Soil,* from *Victorian Women Writer’s Project*). The error message read:

```
In the attribute part of element l, i is not one of the allowed
values
in unnamed entity at line 2122 char 12
In the attribute part of element l, f is not one of the allowed
values
in unnamed entity at line 2125 char 12
```

And at line 2122 – 2125:

```
<l part="i">Why not a sweep?&rdquo;</l>
</lg>
<lg type="para">
<l part="f">The farmer looked, and saw</l>
```

I think that the solution for this problem is simply a matter of defining entities and elements more carefully in the beginning of the document.

Title	tags	Whitespace &	Undefined Entity	Undeclared Elements*
alabama1				
alabama-orig	X(titlePart/LB)			X
alco1			X (OHsagr)	
alco-orig	X(DIV/PB)			X
brait1	Expected name, but got <space>	X		
brait-orig		X		X
brawley1				
brawley-orig	X(titlePart/LB)			X
brown1			X (oelig)	
brown-orig	X(titlePart/LB)			X
catechsl1				
catechsl-orig	X(titlePart/LB)			X
cawe1	Expected name, but got <space>	X		
cawe-orig		X		X
corelli1				
corelli-orig	X(titlePart/LB)			X
ellis1				
ellis-orig	X(P/PB)			X
gild1				
gild-orig	X (item/LB)			X
lanie1				
lanie-orig	X(titlePart/LB)			X
long1	X(revisionDesc/Change)			
long-orig	X(revisionDesc/Change)			X
milla1			X (hellip)	
milla-orig	X(DIV/PB)			X
smith1			X (oelig)	
smith-orig	X(titlePart/LB)			X
ward1			X (adot)	
ward-orig	X(P/PB)			X

Table 6: Documents which were not well-formed

*"Undeclared elements" is solely a problem of those documents which were translated using the original Fekete script. That problem was corrected in the second Viles script.

Of the secondary documents that were not well-formed, the problems were with start/end tags (the correction was discussed in the "well-formed" section above), unidentified entitites (also

discussed in the “well-formed” section above) and the Ampersand *&* Connector and ensuing white space.

In brait1 (William Stanley Braithwaite, *Anthology of magazine verse for 1920 and year book of american poetry* from the *Humanities Text Initiative*), the error report reads:

```
Expected name, but got <space> for entity in unnamed entity at line 136 char 28
```

At line 136:

```
<publisher>Small, Maynard & Company</publisher>
```

The solution to the Ampersand connector problem seems simple: when initially marking up the text, be sure to change all “&” characters to &. And if we want to make doubly sure that the resulting XML output is correct, we could include some sort of find and replace in the third generation perl script to make sure it works.

Title	Well-Formed?	Can IE display it?
alabama1	Y (not valid)	Y
ellis1	Y (not valid)	Y
brawley1	Y (not valid)	Y
gild1	Y (valid?)	Y
catechsl1	Y (valid?)	Y
ward1	N (adot entity)	Y
smith1	N (content model problems, oelig entity)	Y
corelli1	N (doc ends too soon? Unidentified entity? Illegality problems?)	Y
lanie1	N (expected </foreign>, got </l>)	Y
denali…ames-try	N (PCDATA not allowed in unnamed entity, @115)	Y
brown1	N (unidentified oelig entity)	Y
long1	N (expected </change> got </revisionDesc> @ 145)	N
gild-orig	N (expected </LB>, got </ITEM> @ 157)	N
brait1	N (expected name but got <space> for entity @136)	N
cawe1	N (expected name but got <space> for entity @164)	N
milla1	N (hellip entity)	N
alco1	N (OHsagr entity)	N
alabama-orig	N (UNDECLARED ELEMENTS)	N
alco-orig	N (UNDECLARED ELEMENTS)	N
brait-orig	N (UNDECLARED ELEMENTS)	N
brawley-orig	N (UNDECLARED ELEMENTS)	N
brown-orig	N (UNDECLARED ELEMENTS)	N
catechsl-orig	N (UNDECLARED ELEMENTS)	N
cawe-orig	N (UNDECLARED ELEMENTS)	N
corelli-orig	N (UNDECLARED ELEMENTS)	N
ellis-orig	N (UNDECLARED ELEMENTS)	N
lanie-orig	N (UNDECLARED ELEMENTS)	N
long-orig	N (UNDECLARED ELEMENTS)	N
milla-orig	N (UNDECLARED ELEMENTS)	N
smith-orig	N (UNDECLARED ELEMENTS)	N
ward-orig	N (UNDECLARED ELEMENTS)	N

Table 7: Documents and Internet Explorer

I included this table because it is interesting that Internet Explorer will display documents that are not well-formed. It is important to note that Explorer will often try to correct problems it encounters. The non-well-formed documents were displayed until the browser found a problem it could not fix.

The Next Steps

Since the beginning of the TEI project, the need for standardized encoding practices has become even more critical as the need to use and, most importantly, reuse vast amounts of electronic text has dramatically increased for both research and industry. The growing diversity of applications for electronic texts includes natural language processing, scholarly editions, information retrieval, hypertext, electronic publishing, various forms of literary and historical analysis, and lexicography. The central objective of the TEI is to ensure that any text that is created can be used for any number of these applications and for more, as yet not fully understood, purposes. (Thumbnail History). With the advent of XML, those texts which have been encoded in SGML via the TEI scheme can now become available universally over the web. Once we have worked out the kinks in translation, anyone with a computer and an Internet connection will have vast amounts of information at their fingertips. The scripts that I used go a long way in making that happen. However, there are a few problems that need to be addressed before the XML translation is simple and easy.

1. The existing, revised script should be revised again to automate the editorial process at the beginning. (Removing the <!SGML lines and entity references at the beginning, changing DOCTYPE to reference appropriate DTD <!DOCTYPE TEI.2 PUBLIC "-//TEI//DTD TEI Lite XML ver. 1.3//EN" “http://www.uic.edu/orgs/tei/lite/teixlite.dtd,” Removing STYLESPEC and NAVIGATOR lines from DOCTYPE element, Changing ENTITY references of all images from "JPEG” to "jpeg" or from “GIF” to “gif”, Adding entities for copy, ldquo, rdquo, lsquo, rsquo, mdash, and adding ISO references (as per Viles and BonHomme Instructions).

2. When deciding to translate SGML documents to XML format, entity definitions also have to be translated. A script should be written that changes the format from one to the other. This will get rid of most of the problems – from “unidentified entity” to “allowed values”

3. All tags must be closed. SGML allows for non-closed tags but XML does not. A script should be developed to find and replace open tags with tag sets.

4. All documents should be checked to make sure that their “&” connectors are “&(as well as quotations, emdash…)I imagine it’s a pretty simple script.

5. Even when this XML translation is complete, we will still need a stylesheet to view the documents on the web. A universal stylesheet would be too unwieldy, but it would be possible to develop a program (like Pizza Chef) that would allow the user to choose which elements he or she wants in the stylesheet. For example, there could be a stylesheet for each base tag set, as well as stylesheets that would work with the additional tag sets.

Today, the TEI guidelines are is used to encode archival and museum information, classical and medieval literature and language, dictionaries and lexicographies, electronic publishing, English language composition and teaching, historical materials, language corpora, legal texts, literary texts, music historical texts, and religious materials – subjects that are vast in their use and function. If those materials could be made available universally on the web through an application like XML, the “power of the internet” would no longer be a simple marketing ploy. Useful, interesting and powerful information would be available to anyone, anywhere, without reference to time zones or political boundaries. Not only academics would have access to detailed and powerful databases, but normal people sitting at their computers at home would have the power of knowledge literally at their fingertips. But before all of that can happen, the documents have to be encoded so they are compatible with both SGML and XML, and the documents encoded prior to the XML “revolution” must be XML compliant. The work and effort involved will be high, but the payoff is almost limitless. When the kinks get worked out of the translation process, and XML becomes the standard language of the web, digital information in general, and digital libraries in particular will become more powerful, more useful and more significant as learning centers for the population. A new world is upon us!

Bibliography

Bray, Timothy (Bray). *Extensible Markup Language (XML) 1.0. Annotated.* [Online]. Available: http://www.xml.com/axml/target.html [November 28, 1999].

Bryan, Martin. (Bryan). *An introduction to the Extensible Markup Language (XML).* [Online]. Available: http://www.personal.u-net.com/~sgml/xmlintro.htm [November 28, 1999]

Burnard, Lou. (Organization). *Text Encoding for Information Interchange: An introduction to the Text Encoding Initiative: 3. Organization of the TEI Scheme.* [Online]. Available: http://www.hcu.ox.ac.uk/TEI/Papers/J31/ [November 28, 1999]

Burnard, Lou and C.M. Sperberg-McQueen. (TEI Lite Introduction*). TEI Lite: An introduction to Text Encoding for Interchange.* [Online]. Available: http://www.hcu.ox.ac.uk/TEI/Lite/teiu5_en.htm [November 28, 1999].

Burnard, Lou and C. M. Sperberg-McQueen, editors. (Structure). *Guidelines for Text Encoding and Interchange: 3. Structure of the TEI Document Type Definition.* [Online]. Available: http://www.hcu.ox.ac.uk/TEI/P4beta/ST.htm [November 28, 1999]

Burnard, Lou and C. M. Sperberg-McQueen, editors. (Gentle Introduction). *A Gentle Introduction to SGML: 2.4 Defining SGML Document Structures: The DTD.* [Online]. Available: http://www.hcu.ox.ac.uk/TEI/P4beta/SG.htm#SG14 [November 29, 1999]

Fekete, Jean-David. (1999). Perl Script to translate TEI-Lite to TEI-Xlite [Computer programming language]. Paris: Author.

Sperberg-McQueen, C.M.(Construction). *Construction of an XML version of the TEI-DTD.* [Online]. Available: http://www.uic.edu/orgs/tei/ed/edw69.html [November 28, 1999].

Sperberg-McQueen, C.M.(Humanities). *XML and What it Means for Libraries.* [Online]. Available: http://www.uic.edu/~cmsmcq/talks/teidlf1.html [November 30, 1999].

Text Encoding Initiative [TEI]. (Thumbnail History). *A Thumbnail History of the TEI.* [Online]. Available: http://www-tei.uic.edu/orgs/tei/info/hist.html [November 28, 1999]

Text Encoding Initiative [TEI]. (What Is TEI?) *What is the TEI?* [Online]. Available: http://www-tei.uic.edu/orgs/tei/info/teij31/WHAT.htm [November 28, 1999]

Tobin, Richard. (1999?) *XML well-formedness checker and validator* [Online]. Available:

http://www.ltg.ed.ac.uk/%7Erichard/xml-check.html [December 1, 1999]

University of Indiana. Victorian Women's Writers Project. (Victorian) [Online]. Available: http://www.indiana.edu/~letrs/vwwp [November 30, 1999]

University of Michigan. Humanities Text Initiative. (HTI) [Online] Available: http://www.hti.umich.edu/ [November 30, 1999]

University of North Carolina at Chapel Hill. Documenting the American South Project. (DocSouth) [Online]. Available: http://www.metalab.unc.edu/docsouth [November 30, 1999]

Viles, Charles. (1999). Addendum to Fekete Perl Script to translate TEI-Lite to TEI-Xlite. [Computer programming language]. Chapel Hill, NC: Author.

World Wide Web Consortium [W3C]. (1998). *Extensible markup language (XML) 1.0* [Online]. Available: http://www.w3.org/TR/REC-xml [November 28, 1999]

END NOTES

[1] DynaWeb is an application which allows Web client browsers to support full-text, wildcard, Boolean, proximity, context and other complex searching. It's a very powerful tool which allows end-users to locate relevant information which may and can span multiple digital libraries. It's also a very expensive tool, costing nearly $100,000. The final problem with Dynaweb is that by the end of this year, DynaWeb will no longer be produced; meaning there will be no more support for the program, and it will eventually become obsolete.

[2] For a full list of TEI tags, please go to http://etext.lib.virginia.edu/tei/teiquic.html.

[3] For an example of the full TEI-Lite, please go to: http://etext.lib.virginia.edu/tei/teilite-dtd.html.

[4] For a complete list and definition of the tags used in TEI-Lite, please go to: http://etext.lib.virginia.edu/tei/uvatei12.html.

[5] Well-formed XML documents were said to be syntactically correct XML documents. What exactly does that mean (Bray)?

- Begin the XML document with the XML declaration (not required, but strongly recommended).
- A "root" element completely contains the document's content. All other content components must also reside in the "root" element.
- Match start-tags with end-tags. Unlike HTML, XML end-tags are always required.
- If an element does not contain content, the empty element tag may be used.
- Element tags may nest, but never overlap.
- Attribute values must be enclosed in quotes.
- Use & for & and < for <, except in CDATA sections which by definition may contain these delimiter characters.
- The only entity references permitted are & for &, < for <, > for >, ' for ', and " for ".

Valid XML documents are basically well-formed XML documents which include an XML declaration and document type declaration. Valid documents must also adhere to the DTD indicated in the document type declaration. Because entity references may be defined in the DTD, additional entity references may be defined and used in valid documents (Bray).

[6] Having a non-valididated XML document could mean one of three things: the document has no DTD, or there is a DTD for this document, but it's off on a server somewhere and it hasn't been fetched by the client for whatever reason, or there's a DTD available on the serving server, and in fact if checked, the document *would* be valid, but all the program is doing is displaying or indexing it, so the DTD isn't necessary. (Bray) However, if a document is termed "invalid" it means that it doesn't conform to the specified DTD, and will be thrown out of the system.